



PROGRAMMING Expert

This month, Mike James shows you how to use VBScript to create a simple text output object and a complete graphics object that you can draw on using nothing but script commands



MICROSOFT HANDS OUT COURSES AND BOOKS

Until 8th November 2005 Microsoft is offering a range of .NET online programming courses for free. Each lesson includes virtual labs and a skills assessment to help you work out what you need to learn. There are also free books to download from Microsoft's website in return for registering the beta version of Visual Studio 2005. Choose from *Introducing ASP.NET 2.0*, *Introducing Microsoft Visual Basic 2005* and *Writing Secure Code*. To encourage you to move from VB 6 to VB .NET, Microsoft is also offering free copies of *Upgrading Microsoft Visual Basic 6.0 to Microsoft Visual Basic .NET* and *Introducing Visual Basic 2005 for Developers*.

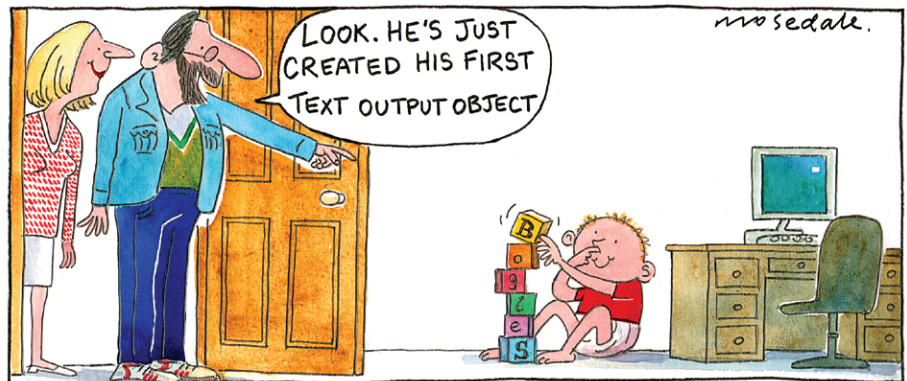
If ASP.NET interests you, download the Developing Microsoft ASP.NET Web Applications Using Visual Studio .NET course, *Microsoft ASP.NET Programming with Microsoft Visual Basic .NET 2003 Step by Step* and *Building Web Solutions with ASP.NET and ADO .NET*. See <http://lab.msdn.microsoft.com/vs2005/learning> for details.

■ ASP.NET FOR LINUX

While .NET is mainly for Windows developers, you can use it for other platforms. Mainsoft has announced a system that allows ASP.NET to work on Linux and other platforms that can run Java. Visual MainWin for J2EE Developer Edition is a Visual Studio .NET plug-in. It's free to download from <http://dev.mainsoft.com> and adds ASP.NET for Linux capabilities to Visual Studio by converting ASP.NET to Java J2EE. Instead of porting the entire .NET system to Linux it compiles Microsoft Intermediate Language (MSIL) to standard Java byte code.

■ VISUAL STUDIO 2005

Microsoft has announced that Visual Studio 2005 and SQL Server 2005 will launch on 7th November, with just one month of 2005 left – thus saving itself the embarrassment of changing the product name. There is likely to be little in it to surprise anyone, however, as both products have been available in beta throughout 2005. See www.microsoft.com.



Windows' free built-in scripting language is useful for everything from system management and web page programming to automating Office applications. We've covered it many times, but it has a few important features missing, notably the ability to create versatile user interfaces for your programs. Its text and graphics output features are minimal.

In *Programming Expert*, Shopper July 2005, we used HTML to create forms that we could use as dialog boxes and even entire interfaces. However, this was fiddly. This month we'll use the object-oriented programming features of VBScript to create interfaces that can be better integrated with your scripts. More specifically, we'll create a simple text output object and a complete graphics object that you can draw on using nothing but script commands. Both can be re-used in any number of scripting projects.

Many people find objects complicated, but they are too useful to ignore. In this project we'll turn a user interface into an object using the VBScript Class facility, thereby wrapping its inner workings in a layer that makes it easy to manipulate using script commands. By turning the interface into an object we can reuse it as many times as we need in a single program. This technique also does away with the need for a separate HTML file to define the interface – as required in our project in *Shopper* July 2005. Instead the HTML we need is created automatically on the fly by the object's constructor.

The techniques you'll learn this month include object-oriented scripting, creating HTML pages dynamically, wrapping user interfaces in objects, creating a text output object, creating vector graphics in HTML (VML) and creating a graphics output object.

A TEXT OUTPUT OBJECT

Text output may not be the most exciting user interface but you really miss the facility when you don't have it. A text output window is excellent for

delivering data on how the script is operating during debugging, testing and maintenance. It's ideal for any output that's not worth designing a custom output form for. We want something low-tech and easy to use, such as Visual Basic's Print command.

We could create a suitable HTML form and load it into Explorer but this separates the code from the interface. In other words, you need to keep track of two files to use the facility. A better but slightly harder way of working is to generate the HTML you need within your code, but this has the disadvantage that you have to remember to call an initialisation routine to create the interface before you use it. An even better way is to create a class that contains all the code you need to create and work an object (the object in this case being a text output box). To create the class you use the VBScript Class command, one of the less well-known Windows script commands. If you open Notepad and type in the script:

```
Class MyClass
    Private MyVar1
    Public MyVar2
    Public Sub MySub
        <some code here>
    End Sub
End Class
```

you create a standard definition of an object of type MyClass. You can then create an object called MyObj of type MyClass, using the command:

```
Set MyObj=New MyClass
```

Think of the class as the recipe for creating objects. Each object is a separate copy of the class containing all its code. Any code or variable declared as Private is internal to the class and cannot be accessed from code outside the object. Anything declared as Public, however, can be accessed by the outside world as a method or a property. For example, with the class definition



given earlier you can write:

```
MyObj.MyVar2= "some data"
MyObj.MySub
```

This sets the value of MyVar2 and runs the subroutine MySub. You can't set MyObj.MyVar1=x because MyVar1 is a private variable. As well as any methods you might define in a class you have:

```
Private Sub Class_Initialize
```

which is called automatically when you create an object based on the class. In the jargon it is called the class constructor. You can also write:

```
Private Sub Class_Terminate
```

which is called automatically when the class is being destroyed. The jargon calls this the class destructor. The constructor and destructor are useful when there's a lot of work to be done in creating and tearing down the class. All this will become clear as we move on to a real example.

PAPER AND PRINT

Start a new script using an ASCII text editor, such as Notepad. First we create a class with a constructor that will create an HTML form. This form will include a single TextArea component ready for use as a printing area. The class is called CPaper and it starts:

```
Class CPaper
```

The constructor does the work of creating our interface. First it creates an instance of Internet Explorer and the Wscript shell:

```
Private Sub Class_Initialize
Set Browser = WScript.CreateObject( _
    "InternetExplorer.Application", _
    "Browser")
Set WshShell = WScript.CreateObject( _
    "WScript.Shell")
```

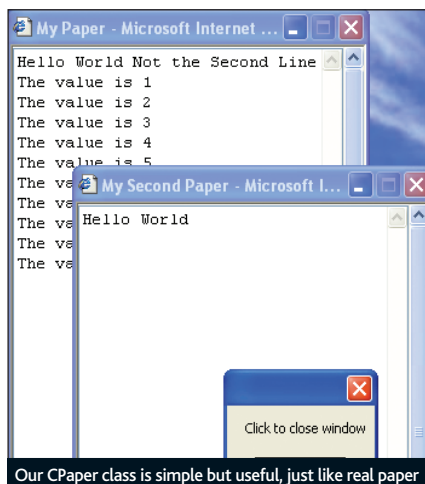
Next it customises the browser to make it look less like a browser:

```
With Browser
.Navigate "about:blank"
.AddressBar = false
.FullScreen = false
.MenuBar = false
.Resizable = false
.StatusBar = false
.ToolBar = false
.height=400
.width=300
End With
```

It makes things easier if we set up some variables that point to objects we'll use a lot:

```
Set Doc=Browser.Document
Set Window=Doc.parentwindow
Set Body=Doc.Body
```

Next we generate the HTML we need for the text box. As this includes double quotes, which tend to be confusing, it is easier to define a 'double quote' variable, called DQ, to use in their place:



```
DQ=CHR(ASC("''"))
textbox="<textarea id=" & DQ _
    & "Text1" & DQ & "></textarea>"
```

The HTML defining the text box can now be stored between the <body> and </body> tags using the command:

```
Body.innerHTML=textbox
```

So that we can refer to the text box in other parts of the program we retrieve a reference to it:

```
set Text1=Doc.all("Text1")
```

We have almost finished. All we do now is position and size the text box, make the browser visible and activate it to make it the top window:

```
Body.LeftMargin=0
Body.TopMargin=0
Text1.style.width=Doc.Body.
ClientWidth
Text1.style.height= _
    Doc.Body.ClientHeight
Browser.Visible=True
WshShell.AppActivate "about:blank"
End Sub
```

So that all subroutines and functions within the object can access the variables, those variables have to be declared just after the Class statement:

```
Private Browser
Private Doc
Private Body
Private Window
Private Text1
```

Now, before running the program, add the following line to the end of the script:

```
End Class
```

and this single line to the start of the script:

```
Set Paper=New CPaper
```

Save the program with the extension VBS, for Visual Basic Script, so that you can execute the file by double-clicking. When you do so you will see an HTML page complete with text box and vertical scroll bar open in a window. This is created without

you having to pre-prepare such an HTML file on your hard disk.

METHODS

All that is left is to add some properties and methods to make the class more useful. We need three basic methods: PrintLN to print text and start a new line; Print to print text without a new line; and Clr to clear the screen. These are surprisingly easy to write. Add the following subroutines before the End Class command:

```
Public Sub Print(Text)
    Text1.value=Text1.value & Text
End Sub

Public Sub PrintLN(Text)
    Print Text
    Text1.value=Text1.value & chr(13)
end sub

Public Sub Clr
    Text1.value=""
End Sub
```

Each one manipulates the value property of the text box using the Text1 variable we set up earlier. Each is declared as public, so these can be used from 'outside' the class as well as 'inside'.

A PROPERTY AND A DESTRUCTOR

A class can define property procedures that allow you to assign properties to your home-grown objects as you would for other objects in Windows. For example, you might want to allow the user to set the title that appears in the top bar of the Text output window. You can do this using a Public Property Let procedure:

```
Public Property Let Title(text)
    Doc.Title=text
End Property
```

Although this looks like a subroutine, we can assign a value to it as if it were a public property of the object.

Finally we'll add a destructor to tidy everything up when we have finished with the object:

```
Private Sub Class_Terminate
    MsgBox("Click to close window")
    Browser.quit
End Sub
```

To try it out, add the following to the start of the program:

```
Set Paper=New CPaper
Paper.Title="My Paper"
Paper.Print "Hello World"
Paper.PrintLN " Not the Second Line"
For i=1 to 10
    Paper.PrintLN "The value is " & i
Next
```

The real power of objects and classes is that if you want another output window you can simply create another instance of the class:

```
Set Paper2=New CPaper
Paper2.Title="My Second Paper"
Paper2.PrintLN "Hello World"
```



You can carry on like this to create as many windows as you need.

MOVING ON TO GRAPHICS

Being able to print text is very useful but it's hardly exciting. HTML and the web are supposed to be all about graphics. You might think we could develop the CPaper class into something we could use to draw graphics using standard HTML, but this isn't possible. Neither basic HTML or the more advanced DHTML have extensive facilities for creating graphics. You can load ready-made bitmaps and perform simple manipulations, but you can't draw a circle or a line. For this we must go beyond HTML.

The internet standard for such graphics is Scalable Vector Graphics (SVG) and, strictly speaking, this is what we should be looking at. It has a rival, though, which is Microsoft's Vector Markup Language (VML). This failed to become a standard and so tends to be overlooked because if you want your web pages to work with any browser, SVG is a better bet. However, VML has several advantages if you are trying to create user interfaces for script. The biggest is that SVG requires you to download a large browser add-in whereas VML is built into Explorer 5 and later. Another advantage is that while SVG graphics are contained in a bounding box, VML graphics can be placed anywhere on the page and mixed with other HTML components. This makes it ideal for our purposes. However, SVG is very similar in design and you will find it quite easy to convert the project to use it if you prefer.

The basics of VML are very simple. You need to add some information at the start of the HTML page and from then on you can use the <v> tag to create graphics. For example:

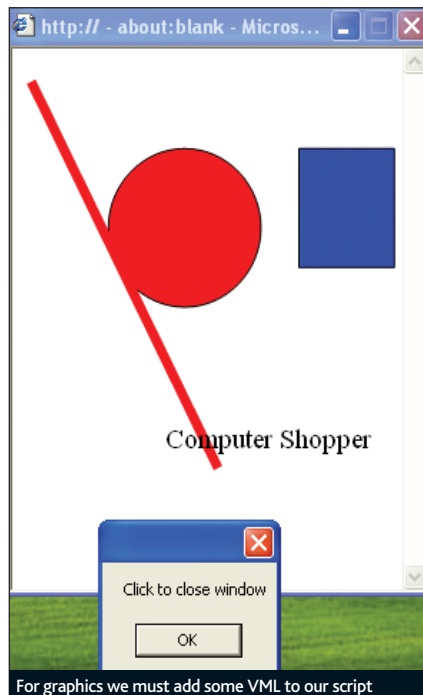
```
<html xmlns:v="urn:schemas-microsoft-com:vml">
<head>
<style>
v\:* { behavior: url(#default#VML); }
</style>
</head>
<body>
<v:oval style='width:100pt;height:75pt' fillcolor="red">
</v:oval>
</body>
</html>
```

If you load this HTML page into a browser you will see a single red ellipse. You can create other shapes by changing oval to rect, roundrect, line and polyline, curve and arc, although the lines and curves need a different set of parameters to position them. You can find out more about VML at www.w3.org/tr/1998/note-vml-19980513.

A CANVAS CLASS

To construct a class that allows us to create general graphics, the process is clear. We have to write a constructor that generates an HTML page like the one above and write methods that add suitable <v> tags. We can't use exactly the same method we used for CPaper to set up the HTML page because we want to change information in the <html> and <head> tags. The simplest solution is to open and write a new program from scratch.

The new class starts in the same way with some variables that we need:



```
Class CCanvas
Private Browser
Private Doc
Private Body
Private Window
Private id
Private DQ
```

The constructor then continues in exactly the same way until the End With instruction. After the End With command we need some new code to write the <html> and <head> tags:

```
DQ=CHR(ASC(" "))
Set Doc=Browser.Document
Set NewDoc = doc.open("text/html",
"replace")
Markup="<html xmlns:v=" & DQ & _
"urn:schemas-microsoft-com:vml" & _
& DQ & "><head>" & _
"<style>v\:* { behavior: url(#default#VML); }" & "</style>" & _
"</head><body></body></html>"
NewDoc.write(Markup)
NewDoc.close()
```

This may look complicated but it simply creates a string Markup containing the HTML shown above. It then creates a new blank document and uses its write method to copy the entire string into the new document. Once the new document is created we can continue as before:

```
Set Window=Doc.parentwindow
Set Body=Doc.Body
Browser.Visible=True
WshShell.AppActivate _
"http:// - about:blank"
id=1
End Sub
```

The window that is created by the doc.open method has a default name of 'http:// - about:blank' rather than just 'about:blank' for reasons

known only to Microsoft. The id variable will be used to keep track of the graphics objects created.

The destructor and the Let Title routines are identical so we won't list them again. The first new method is Oval, which adds an oval object to the body HTML:

```
Public Function Oval(x,y,h,w,c)
Body.innerHTML=Body.innerHTML & _
MakeCmd("oval",x,y,h,w,c,id)
set Oval=doc.all("C" & id)
id=id+1
End Function
```

This calls the MakeCmd subroutine, outlined below, which takes a command name, position, height, width and colour and returns the corresponding HTML for the graphic. The Oval function also returns a reference to the newly created object. The main program can also use this to manipulate the object (as demonstrated later). Part of the reference name is the id variable, which increases by increments so that the next object has a different id. The MakeCmd subroutine uses various other subroutines to build up the HTML piece by piece:

```
Private Function MakeCmd(cmd,x,y,h,w,c,id)
c,id)
MakeCmd="<v:" & cmd & MakeID(id) & _
MakeStyle(x,y,h,w) & _
MakeColour(c) & "> </v:" & _
cmd & ">"
End Function
```

MakeID is simple enough:

```
Private Function MakeID(id)
MakeID=" Id=" & DQ & _
& "C" & id & DQ & " "
End Function
```

The MakeStyle subroutine is a little more complicated as it combines the x, y, h and w parameters into a style string:

```
Private Function MakeStyle(x,y,h,w)
MakeStyle=" style='position: absolute;left:" & _
& x & "pt; top:" & y & _
& "pt; width:" & w & _
& "pt; height:" & h & "pt"'
End Function
```

Finally we have the MakeColour subroutine:

```
Private Function MakeColour(c)
MakeColour="fillcolor=" & DQ & c & _
DQ
End Function
```

After finishing all this with an End Class statement, we can now write a few lines of code to try it out. First, the line:

```
Set Canvas=New CCanvas
```

creates a new Canvas object and:

```
Set ov=Canvas.Oval( _
20,30,40,40,"RGB(255,0,0)")
```




draws a red circle at 20,30. We have specified the colour as an RGB string, but you can also use a valid colour name such as 'red'.

We can use the reference to the new graphics object to modify its properties. For example, to change the fill colour to blue we could use:

```
ov.fillcolor="blue"
```

We can even move the object using:

```
For i=1 To 100
    ov.style.left=i
Next
```

This makes the Canvas object more sophisticated than a simple bitmap and more like a retained mode graphics object. Every graphics object you create keeps its identity and can be manipulated.

Now that we have the Oval method and the functions that support it, adding shapes that use the same parameters is easy. For example, to draw a rectangle we simply add a single new method:

```
Public Function Rect(x,y,h,w,c)
    Body.innerHTML=Body.innerHTML & _
        MakeCmd("rect",x,y,h,w,c,id)
    set Rect=doc.all("C" & id)
    id=id+1
End Function
```

However, this isn't quite the end of the story as other types of shape require more complicated HTML. For example, a line has a start and finish point, a thickness and a colour.

To create this we need some additional helper subroutines, which work much like those above:

```
Public Function Line(x1,y1,x2,y2,t,c)
    Body.innerHTML=Body.innerHTML & _
        MakeLine("line",x1,y1,x2,y2,t,c,
        id)
    set Line=doc.all("C" & id)
    id=id+1
End Function

Private Function MakeLine(cmd,x1,y1,
x2,y2,t,c,id)
    MakeLine="<v:" & cmd & _
        MakeID(id) & MakeFrom(x1,y1) & _
        MakeTo(x2,y2) & _
        MakeStrokeWeight(t) & _
        MakeStrokeColour(c) & _
        "> </v:" & cmd & ">"
End Function

Private Function MakeFrom(x,y)
    MakeFrom=" from=" & MakePoint(x,y)
End Function

Private Function MakeTo(x,y)
    MakeTo=" to=" & MakePoint(x,y)
End Function

Private Function MakePoint(x,y)
    MakePoint=DQ & x & "pt," & y & "pt"
    & DQ
End Function

Private Function MakeStrokeWeight(t)
```

REVIEW BOOK

Beginning Visual Web Programming in C#: From Novice to Professional

RATING ★★★★★

PRICE £22

ISBN 1590593618

PAGES 664

This book claims to be for novice and professional programmers, but you have to know C# to get anything out of it. It is clear about the differences between web and desktop programming and explains how to set up a website and the differences between client- and server-side code.

The pace is slow but it explains everything. However, it doesn't go deep, and you'll need another book if you want to do anything unusual. The authors, Daniel Cazzulino, Victor Garcia Aprea, James Greenwood and Chris Hart, do a good job of speaking with a single voice, but differences of approach are apparent as the book progresses.

This book is about ASP .NET, and there is no point in reading it if you want to use something

```
MakeStrokeWeight=" strokeweight=" &
DQ & t & DQ
End Function

Private Function MakeStrokeColour(c)
    MakeStrokeColour=" strokecolor=" &
DQ & c & DQ
End Function
```

One of the advantages of building programs using lots of subroutines and functions is that you can re-use them in a class to add additional features. For example, the oval and rectangle shapes also support a stroke weight and stroke colour specification and, using the two new Functions, you can add these with almost no additional work.

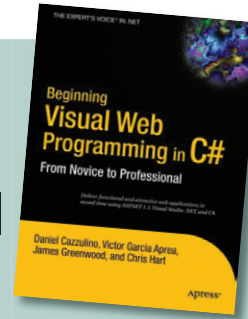
Finally we need two additional methods: a Clr method and a Print method. Clearing the screen is easy, but remember to reset the id count too:

```
Public Sub Clr()
    Body.innerHTML=""
    id=1
End Sub
```

Creating a Print function seems a little more tricky, but you can use the <Div></Div> tags to group any HTML components and support a Style attribute that allows positioning and formatting. So our Print at position xy method simply needs to construct a section of HTML bounded by Div tags:

```
Public Function Print(text,x,y)
    Body.innerHTML=Body.innerHTML & _
        MakePrint(text,x,y,c,id)
    set Print=doc.all("C" & id)
    id=id+1
End Function

Private Function MakePrint(text,x,y,
c,id)
    MakePrint="<Div " & _
```



else. It covers basic server-side coding and ADO .NET as a way of interacting with databases. It also uses Visual Studio .NET almost exclusively. An example case study starts in the early chapters and grows

as the book proceeds. However, you won't be overwhelmed by pages of listings as examples are presented in small chunks with explanations. The book is strong on principles and tells you how something works before showing the code. It also goes beyond programming to discuss setting up a web server, configuration files, security issues, performance tuning and debugging. It even tackles topics such as XML and web services.

This isn't the only book on ASP .NET you will need but if you know C# and want to extend your knowledge to web applications, it's a start.



Explains how things work and why



You need to know C#; doesn't cover the more sophisticated aspects of web programming

```
MakeStyle2(x,y) & _
"> " & text & "</Div>"
End Function

Private Function MakeStyle2(x,y)
    MakeStyle2=" style='position:
absolute;left:" & _
& x & "pt; top:" & y & _
& "pt"'
End Function
```

Now you can clear the drawing area and print text anywhere you like using commands such as:

```
Canvas.Clr
Canvas.Print "Computer Shopper",
80,190
```

There is also a special text box called VML element that we can use to attach text to shapes.

You can carry on adding shapes and drawing commands for hours, but my approach to this sort of class is to start using it in other applications straight away and add extra facilities only when I need them. You can use CCanvas to create as many Canvas objects as you require. It could be very flexible, but for certain projects you might be better off using bitmap graphics. The Canvas drawing commands are all retained within the web page and the whole lot are obeyed every time you make a change. This makes the VML approach to graphics more powerful than that of bitmap graphics but it's not as fast for some applications. **CS**

CONTACT

MIKE JAMES

Mike has written several books on computing and programming and has been a senior lecturer at Teesside University

mike@computershopper.co.uk